autogluon_each_location

October 6, 2023

[3]:
```
import pandas as pd
from darts import TimeSeries
import numpy as np



import warnings
warnings.filterwarnings("ignore")

def fix_datetime(X, name):
    """
    Function to fix and standardize datetime in the given DataFrame.

    Parameters:
    - X: DataFrame to be modified.
    - name: String representing the name of the DataFrame, used for logging.

    Returns:
    - Modified DataFrame with standardized datetime.
    """

    # Convert 'date_forecast' to datetime format and replace original column␣
 ↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')

    # Sort DataFrame by the new datetime column ('ds') and set it as the index
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)

    # Log the shape of the DataFrame before dropping rows with in-between␣
 ↪minutes
    print(f"Shape of {name} before dropping in-between hour rows: ", X.shape)

    # Identify and log gaps in the date sequence
    print(f"HEIHEI: {name} gaps in dates: ", X.index.to_series().diff().dt.
 ↪total_seconds().gt(60*15).sum())
```

```
    print(f"HEIHEI: {name} first gap in dates: ", X[X.index.to_series().diff().
↪dt.total_seconds().gt(60*15)==True].index[:1])

    # Calculate and log the size of each gap in the date sequence
    temp = X.index.to_series().diff().dt.total_seconds()
    if temp.shape[0] > 0:
        print(f"HEIHEI: {name} list of size (in days) of each gap: ", temp[temp.
↪gt(60*15)].values / (60*60*24))

    # temporarily transform into darts time series to fill missing dates
    # get date_calc if date_calc is column in X
    temp_calc = None
    if "date_calc" in X.columns:
        temp_calc = X["date_calc"]
        X.drop(columns=['date_calc'], inplace=True)
    X = TimeSeries.from_dataframe(df=X, freq="15T", fill_missing_dates=True,␣
↪fillna_value=None).pd_dataframe()
    if temp_calc is not None:
        X["date_calc"] = temp_calc

    print(f"HEIHEI: {name} gaps in dates after filling missing dates: ", X.
↪index.to_series().diff().dt.total_seconds().gt(60*15).sum())


    # Drop rows where the minute part of the time is not 0
    X = X[X.index.minute == 0]

    # Log the shape of the DataFrame after dropping rows with in-between minutes
    print(f"Shape of {name} after dropping in-between hour rows: ", X.shape)

    return X



def convert_to_datetime(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_datetime(X_train_observed, "X_train_observed")
    X_train_estimated = fix_datetime(X_train_estimated, "X_train_estimated")
    X_test = fix_datetime(X_test, "X_test")


    X_train_observed["estimated_diff_hours"] = 0
    X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index - pd.
↪to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
    X_test["estimated_diff_hours"] = (X_test.index - pd.
↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600
```

```
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)

    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




# location_map = {
#     "A": 0,
#     "B": 1,
#     "C": 2
# }


def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
 ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
 ↪convert_to_datetime(X_train_observed, X_train_estimated, X_test, y_train)


    # # cast all columns to float64
    # X_train = X_train.astype('float64')
    # X_test = X_test.astype('float64')


    print(f"X_train_observed shape: {X_train_observed.shape}")
    print(f"X_train_estimated shape: {X_train_estimated.shape}")
    print(f"X_test shape: {X_test.shape}")
    print(f"y_train shape: {y_train.shape}")



    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    print("y_train columns: ", y_train.columns)

    # temporarily transform into darts time series to fill missing dates
```

```
  print("Shape of y_train before filling missing dates: ", y_train.shape)
  y_train = TimeSeries.from_dataframe(df=y_train, freq="H",␣
↪fill_missing_dates=True, fillna_value=None).pd_dataframe()
  print("Shape of y_train after filling missing dates: ", y_train.shape)


  # number of gaps in X_train_observed + X_train_estimated before
  print(f"LOOK: Number of gaps in X_train_observed plus number of gaps in␣
↪X_train_estimated before: ", X_train_observed.index.to_series().diff().dt.
↪total_seconds().gt(3600).sum() + X_train_estimated.index.to_series().diff().
↪dt.total_seconds().gt(3600).sum())
  X_train = pd.concat([X_train_observed, X_train_estimated])
  print(f"LOOK: Number of gaps in X_train_observed plus number of gaps in␣
↪X_train_estimated after: ", X_train.index.to_series().diff().dt.
↪total_seconds().gt(3600).sum())
  # print size of gaps in X_train
  temp = X_train.index.to_series().diff().dt.total_seconds()
  if temp.shape[0] > 0:
      print("LOOK: list of size (in days) of each gap: ", temp[temp.gt(3600)].
↪values / (60*60*24))
  print("if the number is bigger after than before that means there is a gap␣
↪in time between the observed and estimated training sets")

  # print info on dates in X_train, and if there are any missing dates
  print("X_train dates info: ", X_train.index.min(), X_train.index.max(),␣
↪X_train.index.max() - X_train.index.min())
  print("X_test dates info: ", X_test.index.min(), X_test.index.max(), X_test.
↪index.max() - X_test.index.min())
  print("y_train dates info: ", y_train.index.min(), y_train.index.max(),␣
↪y_train.index.max() - y_train.index.min())

  # any gaps in dates?
  print("X_train gaps in dates: ", X_train.index.to_series().diff().dt.
↪total_seconds().gt(3600).sum())
  print("X_test gaps in dates: ", X_test.index.to_series().diff().dt.
↪total_seconds().gt(3600).sum())
  print("y_train gaps in dates: ", y_train.index.to_series().diff().dt.
↪total_seconds().gt(3600).sum())

  # temporarily transform into darts time series to fill missing dates
  X_train = TimeSeries.from_dataframe(df=X_train, freq="H",␣
↪fill_missing_dates=True, fillna_value=None).pd_dataframe()
  X_test = TimeSeries.from_dataframe(df=X_test, freq="H",␣
↪fill_missing_dates=True, fillna_value=None).pd_dataframe()
  print("X_train gaps in dates after filling missing dates: ", X_train.index.
↪to_series().diff().dt.total_seconds().gt(3600).sum())
```

```
    print("X_test gaps in dates after filling missing dates: ", X_test.index.
 ↪to_series().diff().dt.total_seconds().gt(3600).sum())



    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    # print Number of missing values in X train
    print("Number of missing values in X_train: ", X_train.isnull().sum().sum())
    print("Number of missing values in X_test: ", X_test.isnull().sum().sum())
    # y_train missing values
    print("Number of missing values in y_train: ", y_train.isnull().sum().sum())
    X_train = pd.merge(X_train, y_train, how="outer", left_index=True,␣
 ↪right_index=True)
    print("Number of missing values in X_train after merging with y_train: ",␣
 ↪X_train.drop(columns=['y']).isnull().sum().sum())



    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test



# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
y_trains = []
# Loop through locations
for loc in locations:
    print("\n\n")
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
```

```
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Concatenate observed and estimated datasets for each location
    #X_train = pd.concat([X_train_estimated, X_train_observed])




    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
 ↪X_test_estimated, y_train, loc)

    print(f"Final shape of X_train for location {loc}: ", X_train.shape)
    print(f"Final shape of X_test for location {loc}: ", X_test.shape)

    # print(y_train.head(), y_train.shape)
    # print(X_train.head(), X_train.shape)
    # print(X_train.head(), X_train.shape)
    # print(type(X_train['y']))

    # Save data to csv
    X_train.to_csv(f'{loc}/X_train.csv', index=True)
    X_test.to_csv(f'{loc}/X_test.csv', index=True)


    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)


# temporary
# X_train["hour"] = X_train.index.hour
# X_train["weekday"] = X_train.index.weekday
# X_train["month"] = X_train.index.month
# X_train["year"] = X_train.index.year

# X_test["hour"] = X_test.index.hour
# X_test["weekday"] = X_test.index.weekday
# X_test["month"] = X_test.index.month
# X_test["year"] = X_test.index.year

print(f"Final shape of X_train: ", X_train.shape)
print(f"Final shape of X_test: ", X_test.shape)
```

```
X_train.dropna(subset=['y'], inplace=True)
X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
Processing location A…
Shape of X_train_observed before dropping in-between hour rows:  (118669, 45)
HEIHEI: X_train_observed gaps in dates:  0
HEIHEI: X_train_observed first gap in dates:  DatetimeIndex([],
dtype='datetime64[ns]', name='ds', freq=None)
HEIHEI: X_train_observed list of size (in days) of each gap:  []
HEIHEI: X_train_observed gaps in dates after filling missing dates:  0
Shape of X_train_observed after dropping in-between hour rows:  (29668, 45)
Shape of X_train_estimated before dropping in-between hour rows:  (17576, 46)
HEIHEI: X_train_estimated gaps in dates:  1
HEIHEI: X_train_estimated first gap in dates:  DatetimeIndex(['2023-01-27'],
dtype='datetime64[ns]', name='ds', freq=None)
HEIHEI: X_train_estimated list of size (in days) of each gap:  [1.01041667]
HEIHEI: X_train_estimated gaps in dates after filling missing dates:  0
Shape of X_train_estimated after dropping in-between hour rows:  (4418, 46)
Shape of X_test before dropping in-between hour rows:  (2880, 46)
HEIHEI: X_test gaps in dates:  17
HEIHEI: X_test first gap in dates:  DatetimeIndex(['2023-05-06'],
dtype='datetime64[ns]', name='ds', freq=None)
HEIHEI: X_test list of size (in days) of each gap:  [4.01041667 7.01041667
3.01041667 1.01041667 1.01041667 1.01041667
 1.01041667 1.01041667 1.01041667 2.01041667 1.01041667 1.01041667
 3.01041667 2.01041667 3.01041667 1.01041667 1.01041667]
HEIHEI: X_test gaps in dates after filling missing dates:  0
Shape of X_test after dropping in-between hour rows:  (1536, 46)
X_train_observed estimated_diff_hours nan:  0
X_train_estimated estimated_diff_hours nan:  24
X_test estimated_diff_hours nan:  816
X_train_observed estimated_diff_hours inf:  0
X_train_estimated estimated_diff_hours inf:  0
X_test estimated_diff_hours inf:  0
```

```
---------------------------------------------------------------------------
IntCastingNaNError                        Traceback (most recent call last)
/Users/jorgensandhaug/Desktop/tdt4173/data/autogluon_each_location.ipynb Cell 1
 ↪line 2
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=202'>203</a>␣
 ↪X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')
```

```
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=204'>205</a> # Concatenate␣
↪observed and estimated datasets for each location
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=205'>206</a> #X_train = pd␣
↪concat([X_train_estimated, X_train_observed])
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=206'>207</a>
  (…)
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=209'>210</a>
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=210'>211</a> # Preprocess␣
↪data
--> <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=211'>212</a> X_train,␣
↪X_test = preprocess_data(X_train_observed, X_train_estimated,␣
↪X_test_estimated, y_train, loc)
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=213'>214</a> print(f"Final␣
↪shape of X_train for location {loc}: ", X_train.shape)
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=214'>215</a> print(f"Final␣
↪shape of X_test for location {loc}: ", X_test.shape)

/Users/jorgensandhaug/Desktop/tdt4173/data/autogluon_each_location.ipynb Cell 1␣
↪line 1
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=107'>108</a> def␣
↪preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
↪location):
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=108'>109</a>     # convert␣
↪to datetime
--> <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=109'>110</a>     ␣
↪X_train_observed, X_train_estimated, X_test, y_train =␣
↪convert_to_datetime(X_train_observed, X_train_estimated, X_test, y_train)
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=112'>113</a>     # # cast␣
↪all columns to float64
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=113'>114</a>     # X_train␣
↪= X_train.astype('float64')
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=114'>115</a>     # X_test ␣
↪X_test.astype('float64')
    <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=117'>118</a>     ␣
↪print(f"X_train_observed shape: {X_train_observed.shape}")
```

```
/Users/jorgensandhaug/Desktop/tdt4173/data/autogluon_each_location.ipynb Cell 1
 ↪line 8
     <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=79'>80</a> print("X_test
 ↪estimated_diff_hours inf: ", np.isinf(X_test["estimated_diff_hours"]).sum())
     <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=80'>81</a> # convert to in
---> <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=81'>82</a>
 ↪X_train_estimated["estimated_diff_hours"] =
 ↪X_train_estimated["estimated_diff_hours"].astype('int64')
     <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=82'>83</a>
 ↪X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].astype('int64 )
     <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#W0sZmlsZQ%3D%3D?line=87'>88</a>
 ↪X_train_estimated.drop(columns=['date_calc'], inplace=True)

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/pandas/core/
 ↪generic.py:6240, in NDFrame.astype(self, dtype, copy, errors)
   6233     results = [
   6234         self.iloc[:, i].astype(dtype, copy=copy)
   6235         for i in range(len(self.columns))
   6236     ]
   6238 else:
   6239     # else, only a single dtype is given
-> 6240     new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   6241     return self._constructor(new_data).__finalize__(self,
 ↪method="astype")
   6243 # GH 33113: handle empty frame or series

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/pandas/core/
 ↪internals/managers.py:448, in BaseBlockManager.astype(self, dtype, copy,
 ↪errors)
    447 def astype(self: T, dtype, copy: bool = False, errors: str = "raise") -
 ↪T:
--> 448     return self.apply("astype", dtype=dtype, copy=copy, errors=errors)

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/pandas/core/
 ↪internals/managers.py:352, in BaseBlockManager.apply(self, f, align_keys,
 ↪ignore_failures, **kwargs)
    350         applied = b.apply(f, **kwargs)
    351     else:
--> 352         applied = getattr(b, f)(**kwargs)
    353 except (TypeError, NotImplementedError):
    354     if not ignore_failures:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/pandas/core/
 ↪internals/blocks.py:526, in Block.astype(self, dtype, copy, errors)
    508 """
```

```
    509 Coerce to the new dtype.
    510
   (…)
    522 Block
    523 """
    524 values = self.values
--> 526 new_values = astype_array_safe(values, dtype, copy=copy, errors=errors)
    528 new_values = maybe_coerce_values(new_values)
    529 newb = self.make_block(new_values)

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/pandas/core/
 ↪dtypes/astype.py:299, in astype_array_safe(values, dtype, copy, errors)
    296     return values.copy()
    298 try:
--> 299     new_values = astype_array(values, dtype, copy=copy)
    300 except (ValueError, TypeError):
    301     # e.g. astype_nansafe can fail on object-dtype of strings
    302     #  trying to convert to float
    303     if errors == "ignore":

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/pandas/core/
 ↪dtypes/astype.py:230, in astype_array(values, dtype, copy)
    227     values = values.astype(dtype, copy=copy)
    229 else:
--> 230     values = astype_nansafe(values, dtype, copy=copy)
    232 # in pandas we don't store numpy str dtypes, so convert to object
    233 if isinstance(dtype, np.dtype) and issubclass(values.dtype.type, str):

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/pandas/core/
 ↪dtypes/astype.py:140, in astype_nansafe(arr, dtype, copy, skipna)
    137     raise TypeError(f"cannot astype a timedelta from [{arr.dtype}] to␣
 ↪[{dtype}]")
    139 elif np.issubdtype(arr.dtype, np.floating) and is_integer_dtype(dtype):
--> 140     return _astype_float_to_int_nansafe(arr, dtype, copy)
    142 elif is_object_dtype(arr.dtype):
    143
    144     # if we have a datetime/timedelta array of objects
    145     # then coerce to a proper dtype and recall astype_nansafe
    147     if is_datetime64_dtype(dtype):

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/pandas/core/
 ↪dtypes/astype.py:182, in _astype_float_to_int_nansafe(values, dtype, copy)
    178 """
    179 astype with a check preventing converting NaN to an meaningless integer␣
 ↪value.
    180 """
    181 if not np.isfinite(values).all():
--> 182     raise IntCastingNaNError(
```

```
    183         "Cannot convert non-finite values (NA or inf) to integer"
    184     )
    185 if dtype.kind == "u":
    186     # GH#45151
    187     if not (values >= 0).all():

IntCastingNaNError: Cannot convert non-finite values (NA or inf) to integer
```

[ ]:
```
import pandas as pd

df = X_train.copy()
test_df = X_test.copy()


# add sin and cos of sun_elevation:d and sun_azimuth:d
df['sin_sun_elevation'] = np.sin(np.deg2rad(df['sun_elevation:d']))

test_df['sin_sun_elevation'] = np.sin(np.deg2rad(test_df['sun_elevation:d']))

# add global_rad_1h:J = diffuse_rad_1h:J + direct_rad_1h:J
df['global_rad_1h:J'] = df['diffuse_rad_1h:J'] + df['direct_rad_1h:J']
test_df['global_rad_1h:J'] = test_df['diffuse_rad_1h:J'] +␣
 ↪test_df['direct_rad_1h:J']

# dew_or_rime:idx, Change this to one variable for is_dew and one variable for␣
 ↪is_rime (dew:1, rime:-1)
df['is_dew'] = df['dew_or_rime:idx'].apply(lambda x: 1 if x == 1 else 0)
df['is_rime'] = df['dew_or_rime:idx'].apply(lambda x: 1 if x == -1 else 0)

test_df['is_dew'] = test_df['dew_or_rime:idx'].apply(lambda x: 1 if x == 1 else␣
 ↪0)
test_df['is_rime'] = test_df['dew_or_rime:idx'].apply(lambda x: 1 if x == -1␣
 ↪else 0)


EXOGENOUS = [
    'estimated_diff_hours',
    "absolute_humidity_2m:gm3",
    "air_density_2m:kgm3",
    "dew_point_2m:K",
    "diffuse_rad_1h:J",
    "direct_rad_1h:J",
    "effective_cloud_cover:p",
    "fresh_snow_1h:cm",
    "snow_depth:cm",
    "sun_elevation:d",
```

```
    "sun_azimuth:d",
    "t_1000hPa:K",
    "visibility:m",
    "wind_speed_10m:ms",
    "is_dew",
    "is_rime",
    "sin_sun_elevation",
    "global_rad_1h:J",
    ]
#additional_features_for_testing =

df = df[EXOGENOUS + ["y", "location"]]
test_df = test_df[EXOGENOUS+ ["location"]]

# save to X_train_feature_engineered.csv
df.to_csv('X_train_feature_engineered.csv', index=True)
test_df.to_csv('X_test_feature_engineered.csv', index=True)
```

1 Starting

[ ]:
```
import os
# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
 ↪filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'
```

```
Last submission number: 70
Now creating submission number: 71
```

[ ]:
```
from autogluon.tabular import TabularDataset, TabularPredictor
train_data = TabularDataset('X_train_raw.csv')
train_data.drop(columns=['ds'], inplace=True)

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60
presets = 'best_quality'
```

```
Loaded data from: X_train_raw.csv | Columns = 53 / 53 | Rows = 93024 -> 93024
```

[ ]:
```
predictors = [None, None, None]
```

[ ]:
```
loc = "A"
print(f"Training model for location {loc}...")
predictor = TabularPredictor(label=label, eval_metric=metric,␣
 ↪path=f"AutogluonModels/{new_filename}_{loc}").
 ↪fit(train_data[train_data["location"] == loc], time_limit=time_limit,␣
 ↪presets=presets)
predictors[0] = predictor
```

```
Warning: path already exists! This predictor may overwrite an existing
predictor! path="AutogluonModels/submission_71_A"
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training … Time limit = 240s
AutoGluon will save models to "AutogluonModels/submission_71_A/"
AutoGluon Version:  0.8.1
Python Version:     3.10.12
Operating System:   Darwin
Platform Machine:   arm64
Platform Version:   Darwin Kernel Version 22.1.0: Sun Oct  9 20:15:09 PDT 2022;
root:xnu-8792.41.9~2/RELEASE_ARM64_T6000
Disk Space Avail:   1.10 GB / 494.38 GB (0.2%)
        WARNING: Available disk space is low and there is a risk that AutoGluon
will run out of disk during fit, causing an exception.
        We recommend a minimum available disk space of 10 GB, and large datasets
may require more.
Train Data Rows:    34085
Train Data Columns: 51
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 630.59471,
1165.90242)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    5142.59 MB
        Train Data (Original)  Memory Usage: 15.61 MB (0.3% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 3 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
```

```
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…

Training model for location A…

        Useless Original Features (Count: 1): ['location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Unused Original Features (Count: 1): ['snow_drift:idx']
                These features were not used to generate any of the output
features. Add a feature generator compatible with these features to utilize
them.
                Features can also be unused if they carry very little
information, such as being categorical but having almost entirely unique values
or being duplicates of other features.
                These features do not need to be present at inference time.
                ('float', []) : 1 | ['snow_drift:idx']
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 45 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  4 | ['hour', 'weekday', 'month', 'year']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 43 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])       :  4 | ['hour', 'weekday', 'month', 'year']
                ('int', ['bool']) :  2 | ['elevation:m', 'snow_density:kgm3']
        0.3s = Fit runtime
        49 features in original data used to generate 49 features in processed
data.
        Train Data (Processed) Memory Usage: 12.88 MB (0.3% of available memory)
Data preprocessing and feature engineering runtime = 0.29s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
```

```
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) …
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 159.77s of the
239.71s of remaining time.
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/Users/jorgensandhaug/Desktop/tdt4173/data/autogluon_each_location.ipynb Cell 8
 ↪line 3
      <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#W6sZmlsZQ%3D%3D?line=0'>1</a> loc = "A"
      <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#W6sZmlsZQ%3D%3D?line=1'>2</a> print(f"Training␣
 ↪model for location {loc}…")
----> <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#W6sZmlsZQ%3D%3D?line=2'>3</a> predictor =␣
 ↪TabularPredictor(label=label, eval_metric=metric, path=f"AutogluonModels/
 ↪{new_filename}_{loc}").fit(train_data[train_data["location"] == loc],␣
 ↪time_limit=time_limit, presets=presets)
      <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#W6sZmlsZQ%3D%3D?line=3'>4</a> predictors[0] =␣
 ↪predictor

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor
 ↪utils/decorators.py:31, in unpack.<locals>._unpack_inner.<locals>._call(*args␣
 ↪**kwargs)
     28 @functools.wraps(f)
     29 def _call(*args, **kwargs):
     30     gargs, gkwargs = g(*other_args, *args, **kwargs)
---> 31     return f(*gargs, **gkwargs)
```

```
File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/
 ↪tabular/predictor/predictor.py:986, in TabularPredictor.fit(self, train_data,␣
 ↪tuning_data, time_limit, presets, hyperparameters, feature_metadata,␣
 ↪infer_limit, infer_limit_batch_size, fit_weighted_ensemble,␣
 ↪calibrate_decision_threshold, num_cpus, num_gpus, **kwargs)
    984     aux_kwargs["fit_weighted_ensemble"] = False
    985 self.save(silent=True)  # Save predictor to disk to enable prediction␣
 ↪and training after interrupt
--> 986 self._learner.fit(
    987     X=train_data,
    988     X_val=tuning_data,
    989     X_unlabeled=unlabeled_data,
    990     holdout_frac=holdout_frac,
    991     num_bag_folds=num_bag_folds,
    992     num_bag_sets=num_bag_sets,
    993     num_stack_levels=num_stack_levels,
    994     hyperparameters=hyperparameters,
    995     core_kwargs=core_kwargs,
    996     aux_kwargs=aux_kwargs,
    997     time_limit=time_limit,
    998     infer_limit=infer_limit,
    999     infer_limit_batch_size=infer_limit_batch_size,
   1000     verbosity=verbosity,
   1001     use_bag_holdout=use_bag_holdout,
   1002 )
   1003 self._set_post_fit_vars()
   1005 self._post_fit(
   1006     keep_only_best=kwargs["keep_only_best"],
   1007     refit_full=kwargs["refit_full"],
   (…)
   1012     infer_limit=infer_limit,
   1013 )

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/
 ↪tabular/learner/abstract_learner.py:159, in AbstractTabularLearner.fit(self,␣
 ↪X, X_val, **kwargs)
    157     raise AssertionError("Learner is already fit.")
    158 self._validate_fit_input(X=X, X_val=X_val, **kwargs)
--> 159 return self._fit(X=X, X_val=X_val, **kwargs)

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/
 ↪tabular/learner/default_learner.py:157, in DefaultLearner._fit(self, X, X_val␣
 ↪X_unlabeled, holdout_frac, num_bag_folds, num_bag_sets, time_limit,␣
 ↪infer_limit, infer_limit_batch_size, verbosity, **trainer_fit_kwargs)
    154     self.eval_metric = trainer.eval_metric
    156 self.save()
--> 157 trainer.fit(
    158     X=X,
    159     y=y,
```

```
    160     X_val=X_val,
    161     y_val=y_val,
    162     X_unlabeled=X_unlabeled,
    163     holdout_frac=holdout_frac,
    164     time_limit=time_limit_trainer,
    165     infer_limit=infer_limit,
    166     infer_limit_batch_size=infer_limit_batch_size,
    167     groups=groups,
    168     **trainer_fit_kwargs,
    169 )
    170 self.save_trainer(trainer=trainer)
    171 time_end = time.time()

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/
 ↪tabular/trainer/auto_trainer.py:114, in AutoTrainer.fit(self, X, y,␣
 ↪hyperparameters, X_val, y_val, X_unlabeled, holdout_frac, num_stack_levels,␣
 ↪core_kwargs, aux_kwargs, time_limit, infer_limit, infer_limit_batch_size,␣
 ↪use_bag_holdout, groups, **kwargs)
    111 log_str += "}"
    112 logger.log(20, log_str)
--> 114 self._train_multi_and_ensemble(
    115     X=X,
    116     y=y,
    117     X_val=X_val,
    118     y_val=y_val,
    119     X_unlabeled=X_unlabeled,
    120     hyperparameters=hyperparameters,
    121     num_stack_levels=num_stack_levels,
    122     time_limit=time_limit,
    123     core_kwargs=core_kwargs,
    124     aux_kwargs=aux_kwargs,
    125     infer_limit=infer_limit,
    126     infer_limit_batch_size=infer_limit_batch_size,
    127     groups=groups,
    128 )

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:2371, in AbstractTrainer.
 ↪_train_multi_and_ensemble(self, X, y, X_val, y_val, hyperparameters,␣
 ↪X_unlabeled, num_stack_levels, time_limit, groups, **kwargs)
   2369     self._num_rows_val = len(X_val)
   2370 self._num_cols_train = len(list(X.columns))
-> 2371 model_names_fit = self.train_multi_levels(
   2372     X,
   2373     y,
   2374     hyperparameters=hyperparameters,
   2375     X_val=X_val,
   2376     y_val=y_val,
   2377     X_unlabeled=X_unlabeled,
```

```
   2378     level_start=1,
   2379     level_end=num_stack_levels + 1,
   2380     time_limit=time_limit,
   2381     **kwargs,
   2382 )
   2383 if len(self.get_model_names()) == 0:
   2384     raise ValueError("AutoGluon did not successfully train any models")

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:395, in AbstractTrainer.train_multi_levels(self,␣
 ↪X, y, hyperparameters, X_val, y_val, X_unlabeled, base_model_names,␣
 ↪core_kwargs, aux_kwargs, level_start, level_end, time_limit, name_suffix,␣
 ↪relative_stack, level_time_modifier, infer_limit, infer_limit_batch_size)
    393         core_kwargs_level["time_limit"] = core_kwargs_level.
 ↪get("time_limit", time_limit_core)
    394         aux_kwargs_level["time_limit"] = aux_kwargs_level.
 ↪get("time_limit", time_limit_aux)
--> 395     base_model_names, aux_models = self.stack_new_level(
    396         X=X,
    397         y=y,
    398         X_val=X_val,
    399         y_val=y_val,
    400         X_unlabeled=X_unlabeled,
    401         models=hyperparameters,
    402         level=level,
    403         base_model_names=base_model_names,
    404         core_kwargs=core_kwargs_level,
    405         aux_kwargs=aux_kwargs_level,
    406         name_suffix=name_suffix,
    407         infer_limit=infer_limit,
    408         infer_limit_batch_size=infer_limit_batch_size,
    409     )
    410     model_names_fit += base_model_names + aux_models
    411 if self.model_best is None and len(model_names_fit) != 0:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:539, in AbstractTrainer.stack_new_level(self, X,␣
 ↪y, models, X_val, y_val, X_unlabeled, level, base_model_names, core_kwargs,␣
 ↪aux_kwargs, name_suffix, infer_limit, infer_limit_batch_size)
    537     core_kwargs["name_suffix"] = core_kwargs.get("name_suffix", "") +␣
 ↪name_suffix
    538     aux_kwargs["name_suffix"] = aux_kwargs.get("name_suffix", "") +␣
 ↪name_suffix
--> 539 core_models = self.stack_new_level_core(
    540     X=X,
    541     y=y,
    542     X_val=X_val,
    543     y_val=y_val,
    544     X_unlabeled=X_unlabeled,
```

```
    545     models=models,
    546     level=level,
    547     infer_limit=infer_limit,
    548     infer_limit_batch_size=infer_limit_batch_size,
    549     base_model_names=base_model_names,
    550     **core_kwargs,
    551 )
    553 if X_val is None:
    554     aux_models = self.stack_new_level_aux(
    555         X=X, y=y, base_model_names=core_models, level=level + 1,␣
 ↪infer_limit=infer_limit, infer_limit_batch_size=infer_limit_batch_size,␣
 ↪**aux_kwargs
    556     )

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:673, in AbstractTrainer.stack_new_level_core(self ␣
 ↪X, y, models, X_val, y_val, X_unlabeled, level, base_model_names, stack_name, ,
 ↪ag_args, ag_args_fit, ag_args_ensemble, included_model_types,␣
 ↪excluded_model_types, ensemble_type, name_suffix, get_models_func, refit_full ␣
 ↪infer_limit, infer_limit_batch_size, **kwargs)
    670 fit_kwargs = dict(num_classes=self.num_classes)
    672 # FIXME: TODO: v0.1 X_unlabeled isn't cached so it won't be available␣
 ↪during refit_full or fit_extra.
--> 673 return self._train_multi(
    674     X=X_init,
    675     y=y,
    676     X_val=X_val,
    677     y_val=y_val,
    678     X_unlabeled=X_unlabeled,
    679     models=models,
    680     level=level,
    681     stack_name=stack_name,
    682     compute_score=compute_score,
    683     fit_kwargs=fit_kwargs,
    684     **kwargs,
    685 )

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:2321, in AbstractTrainer._train_multi(self, X, y, ,
 ↪models, hyperparameter_tune_kwargs, feature_prune_kwargs, k_fold, n_repeats,␣
 ↪n_repeat_start, time_limit, **kwargs)
   2319 if n_repeat_start == 0:
   2320     time_start = time.time()
-> 2321     model_names_trained = self._train_multi_initial(
   2322         X=X,
   2323         y=y,
   2324         models=models,
   2325         k_fold=k_fold,
   2326         n_repeats=n_repeats_initial,
   2327         hyperparameter_tune_kwargs=hyperparameter_tune_kwargs,
```

```
   2328         feature_prune_kwargs=feature_prune_kwargs,
   2329         time_limit=time_limit,
   2330         **kwargs,
   2331     )
   2332     n_repeat_start = n_repeats_initial
   2333     if time_limit is not None:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:2170, in AbstractTrainer.
 ↪_train_multi_initial(self, X, y, models, k_fold, n_repeats,␣
 ↪hyperparameter_tune_kwargs, time_limit, feature_prune_kwargs, **kwargs)
   2168 else:
   2169     time_ratio = hpo_time_ratio if hpo_enabled else 1
-> 2170     models = self._train_multi_fold(
   2171         models=models,
   2172         hyperparameter_tune_kwargs=hyperparameter_tune_kwargs,
   2173         k_fold_start=0,
   2174         k_fold_end=k_fold,
   2175         n_repeats=n_repeats,
   2176         n_repeat_start=0,
   2177         time_limit=time_limit,
   2178         time_split=time_split,
   2179         time_ratio=time_ratio,
   2180         **fit_args,
   2181     )
   2183 multi_fold_time_elapsed = time.time() - multi_fold_time_start
   2184 if time_limit is not None:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:2278, in AbstractTrainer._train_multi_fold(self,␣
 ↪X, y, models, time_limit, time_split, time_ratio, hyperparameter_tune_kwargs,␣
 ↪**kwargs)
   2276         time_start_model = time.time()
   2277         time_left = time_limit - (time_start_model - time_start)
-> 2278 model_name_trained_lst = self._train_single_full(
   2279     X, y, model, time_limit=time_left,␣
 ↪hyperparameter_tune_kwargs=hyperparameter_tune_kwargs_model, **kwargs
   2280 )
   2282 if self.low_memory:
   2283     del model

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:2051, in AbstractTrainer._train_single_full(self,␣
 ↪X, y, model, X_unlabeled, X_val, y_val, X_pseudo, y_pseudo, feature_prune,␣
 ↪hyperparameter_tune_kwargs, stack_name, k_fold, k_fold_start, k_fold_end,␣
 ↪n_repeats, n_repeat_start, level, time_limit, fit_kwargs, compute_score,␣
 ↪total_resources, **kwargs)
   2047         bagged_model_fit_kwargs = self._get_bagged_model_fit_kwargs(
   2048             k_fold=k_fold, k_fold_start=k_fold_start,␣
 ↪k_fold_end=k_fold_end, n_repeats=n_repeats, n_repeat_start=n_repeat_start
```

```
   2049         )
   2050         model_fit_kwargs.update(bagged_model_fit_kwargs)
-> 2051     model_names_trained = self._train_and_save(
   2052         X=X,
   2053         y=y,
   2054         model=model,
   2055         X_val=X_val,
   2056         y_val=y_val,
   2057         X_unlabeled=X_unlabeled,
   2058         stack_name=stack_name,
   2059         level=level,
   2060         compute_score=compute_score,
   2061         total_resources=total_resources,
   2062         **model_fit_kwargs,
   2063     )
   2064 self.save()
   2065 return model_names_trained

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:1733, in AbstractTrainer._train_and_save(self, X, 
 ↪y, model, X_val, y_val, stack_name, level, compute_score, total_resources,␣
 ↪**model_fit_kwargs)
   1731     model = self._train_single(X_w_pseudo, y_w_pseudo, model, X_val,␣
 ↪y_val, **model_fit_kwargs)
   1732 else:
-> 1733     model = self._train_single(X, y, model, X_val, y_val,␣
 ↪total_resources=total_resources, **model_fit_kwargs)
   1735 fit_end_time = time.time()
   1736 if self.weight_evaluation:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:1684, in AbstractTrainer._train_single(self, X, y ␣
 ↪model, X_val, y_val, total_resources, **model_fit_kwargs)
   1679 def _train_single(self, X, y, model: AbstractModel, X_val=None,␣
 ↪y_val=None, total_resources=None, **model_fit_kwargs) -> AbstractModel:
   1680     """
   1681     Trains model but does not add the trained model to this Trainer.
   1682     Returns trained model object.
   1683     """
-> 1684     model = model.fit(X=X, y=y, X_val=X_val, y_val=y_val,␣
 ↪total_resources=total_resources, **model_fit_kwargs)
   1685     return model

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪models/abstract/abstract_model.py:829, in AbstractModel.fit(self, **kwargs)
    827 self.validate_fit_resources(**kwargs)
    828 self._validate_fit_memory_usage(**kwargs)
--> 829 out = self._fit(**kwargs)
```

```
    830 if out is None:
    831     out = self

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪models/ensemble/stacker_ensemble_model.py:169, in StackerEnsembleModel.
 ↪_fit(self, X, y, compute_base_preds, time_limit, **kwargs)
    167 if time_limit is not None:
    168     time_limit = time_limit - (time.time() - start_time)
--> 169 return super()._fit(X=X, y=y, time_limit=time_limit, **kwargs)

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪models/ensemble/bagged_ensemble_model.py:250, in BaggedEnsembleModel.
 ↪_fit(self, X, y, X_val, y_val, X_pseudo, y_pseudo, k_fold, k_fold_start,␣
 ↪k_fold_end, n_repeats, n_repeat_start, groups, _skip_oof, **kwargs)
    248 save_bag_folds = self.params.get("save_bag_folds", True)
    249 if k_fold == 1:
--> 250     self._fit_single(X=X, y=y, model_base=model_base,␣
 ↪use_child_oof=use_child_oof, skip_oof=_skip_oof, **kwargs)
    251     return self
    252 else:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪models/ensemble/bagged_ensemble_model.py:400, in BaggedEnsembleModel.
 ↪_fit_single(self, X, y, model_base, use_child_oof, time_limit, skip_oof,␣
 ↪**kwargs)
    398 X_sample = X.sample(n=n_sample)
    399 time_start_predict = time.time()
--> 400 model_base.predict_proba(X_sample)
    401 time_predict_frac = time.time() - time_start_predict
    402 time_predict_estimate = time_predict_frac / frac

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪models/abstract/abstract_model.py:931, in AbstractModel.predict_proba(self, X␣
 ↪normalize, **kwargs)
    929 if normalize is None:
    930     normalize = self.normalize_pred_probas
--> 931 y_pred_proba = self._predict_proba(X=X, **kwargs)
    932 if normalize:
    933     y_pred_proba = normalize_pred_probas(y_pred_proba, self.problem_typ )

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪models/abstract/abstract_model.py:946, in AbstractModel._predict_proba(self,␣
 ↪X, **kwargs)
    943 X = self.preprocess(X, **kwargs)
    945 if self.problem_type in [REGRESSION, QUANTILE]:
--> 946     y_pred = self.model.predict(X)
    947     return y_pred
    949 y_pred_proba = self.model.predict_proba(X)
```

```
File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/sklearn/
 ↪neighbors/_regression.py:236, in KNeighborsRegressor.predict(self, X)
    220 """Predict the target for the provided data.
    221
    222 Parameters
   (…)
    231     Target values.
    232 """
    233 if self.weights == "uniform":
    234     # In that case, we do not need the distances to perform
    235     # the weighting so we do not compute them.
--> 236     neigh_ind = self.kneighbors(X, return_distance=False)
    237     neigh_dist = None
    238 else:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/sklearn/
 ↪neighbors/_base.py:824, in KNeighborsMixin.kneighbors(self, X, n_neighbors,␣
 ↪return_distance)
    817 use_pairwise_distances_reductions = (
    818     self._fit_method == "brute"
    819     and ArgKmin.is_usable_for(
    820         X if X is not None else self._fit_X, self._fit_X, self.
 ↪effective_metric_
    821     )
    822 )
    823 if use_pairwise_distances_reductions:
--> 824     results = ArgKmin.compute(
    825         X=X,
    826         Y=self._fit_X,
    827         k=n_neighbors,
    828         metric=self.effective_metric_,
    829         metric_kwargs=self.effective_metric_params_,
    830         strategy="auto",
    831         return_distance=return_distance,
    832     )
    834 elif (
    835     self._fit_method == "brute" and self.metric == "precomputed" and␣
 ↪issparse(X)
    836 ):
    837     results = _kneighbors_from_graph(
    838         X, n_neighbors=n_neighbors, return_distance=return_distance
    839     )

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/sklearn/
 ↪metrics/_pairwise_distances_reduction/_dispatcher.py:289, in ArgKmin.
 ↪compute(cls, X, Y, k, metric, chunk_size, metric_kwargs, strategy,␣
 ↪return_distance)
    277     return ArgKmin64.compute(
```

```
    278         X=X,
    279         Y=Y,
   (…)
    285         return_distance=return_distance,
    286     )
    288 if X.dtype == Y.dtype == np.float32:
--> 289     return ArgKmin32.compute(
    290         X=X,
    291         Y=Y,
    292         k=k,
    293         metric=metric,
    294         chunk_size=chunk_size,
    295         metric_kwargs=metric_kwargs,
    296         strategy=strategy,
    297         return_distance=return_distance,
    298     )
    300 raise ValueError(
    301     "Only float64 or float32 datasets pairs are supported at this time, "
    302     f"got: X.dtype={X.dtype} and Y.dtype={Y.dtype}."
    303 )

File sklearn/metrics/_pairwise_distances_reduction/_argkmin.pyx:584, in sklearn
 ↪metrics._pairwise_distances_reduction._argkmin.ArgKmin32.compute()

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/threadpoolctl
 ↪py:440, in _ThreadpoolLimiter.__exit__(self, type, value, traceback)
    437 def __enter__(self):
    438     return self
--> 440 def __exit__(self, type, value, traceback):
    441     self.restore_original_limits()
    443 @classmethod
    444 def wrap(cls, controller, *, limits=None, user_api=None):

KeyboardInterrupt:
```

[ ]:
```
loc = "B"
print(f"Training model for location {loc}...")
predictor = TabularPredictor(label=label, eval_metric=metric,␣
 ↪path=f"AutogluonModels/{new_filename}_{loc}").
 ↪fit(train_data[train_data["location"] == loc], time_limit=time_limit,␣
 ↪presets=presets)
predictors[1] = predictor
```

```
Warning: path already exists! This predictor may overwrite an existing
predictor! path="AutogluonModels/submission_71_B"
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8,
```

```
num_bag_sets=20
Beginning AutoGluon training … Time limit = 240s
AutoGluon will save models to "AutogluonModels/submission_71_B/"
AutoGluon Version:  0.8.1
Python Version:     3.10.12
Operating System:   Darwin
Platform Machine:   arm64
Platform Version:   Darwin Kernel Version 22.1.0: Sun Oct  9 20:15:09 PDT 2022;
root:xnu-8792.41.9~2/RELEASE_ARM64_T6000
Disk Space Avail:   1.07 GB / 494.38 GB (0.2%)
        WARNING: Available disk space is low and there is a risk that AutoGluon
will run out of disk during fit, causing an exception.
        We recommend a minimum available disk space of 10 GB, and large datasets
may require more.
Train Data Rows:    32844
Train Data Columns: 48
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, -0.0, 96.82478, 193.94649)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    4767.44 MB
        Train Data (Original)  Memory Usage: 16.49 MB (0.3% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
                Fitting DatetimeFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…

Training model for location B…

        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 1): ['location']
                These features carry no predictive signal and should be manually
investigated.
```

```
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', [])                       : 46 |
['absolute_humidity_2m:gm3', 'air_density_2m:kgm3', 'ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', …]
                ('object', ['datetime_as_object']) :  1 | ['ds']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])               : 44 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool'])           :  2 | ['elevation:m',
'snow_density:kgm3']
                ('int', ['datetime_as_int']) :  5 | ['ds', 'ds.year',
'ds.month', 'ds.day', 'ds.dayofweek']
        0.2s = Fit runtime
        47 features in original data used to generate 51 features in processed
data.
        Train Data (Processed) Memory Usage: 12.94 MB (0.3% of available memory)
Data preprocessing and feature engineering runtime = 0.25s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) …
```

```
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 159.8s of the
239.75s of remaining time.
        Not enough time to generate out-of-fold predictions for model. Estimated
time required was 342.02s compared to 207.7s of available time.
        Time limit exceeded… Skipping KNeighborsUnif_BAG_L1.
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 154.54s of the
234.49s of remaining time.
        Not enough time to generate out-of-fold predictions for model. Estimated
time required was 268.62s compared to 200.86s of available time.
        Time limit exceeded… Skipping KNeighborsDist_BAG_L1.
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 150.38s of the
230.34s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -22.6532         = Validation score   (-mean_absolute_error)
        34.41s   = Training   runtime
        72.09s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 102.96s of the
182.91s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/Users/jorgensandhaug/Desktop/tdt4173/data/autogluon_each_location.ipynb Cell 8
 ↪line 3
      <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#X10sZmlsZQ%3D%3D?line=0'>1</a> loc = "B"
      <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#X10sZmlsZQ%3D%3D?line=1'>2</a> print(f"Training
 ↪model for location {loc}…")
----> <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#X10sZmlsZQ%3D%3D?line=2'>3</a> predictor =␣
 ↪TabularPredictor(label=label, eval_metric=metric, path=f"AutogluonModels/
 ↪{new_filename}_{loc}").fit(train_data[train_data["location"] == loc],␣
 ↪time_limit=time_limit, presets=presets)
      <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#X10sZmlsZQ%3D%3D?line=3'>4</a> predictors[1] =␣
 ↪predictor

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor/
 ↪utils/decorators.py:31, in unpack.<locals>._unpack_inner.<locals>._call(*args␣
 ↪**kwargs)
     28 @functools.wraps(f)
     29 def _call(*args, **kwargs):
     30     gargs, gkwargs = g(*other_args, *args, **kwargs)
---> 31     return f(*gargs, **gkwargs)
```

```
File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/
 ↪tabular/predictor/predictor.py:986, in TabularPredictor.fit(self, train_data,␣
 ↪tuning_data, time_limit, presets, hyperparameters, feature_metadata,␣
 ↪infer_limit, infer_limit_batch_size, fit_weighted_ensemble,␣
 ↪calibrate_decision_threshold, num_cpus, num_gpus, **kwargs)
    984     aux_kwargs["fit_weighted_ensemble"] = False
    985 self.save(silent=True)  # Save predictor to disk to enable prediction␣
 ↪and training after interrupt
--> 986 self._learner.fit(
    987     X=train_data,
    988     X_val=tuning_data,
    989     X_unlabeled=unlabeled_data,
    990     holdout_frac=holdout_frac,
    991     num_bag_folds=num_bag_folds,
    992     num_bag_sets=num_bag_sets,
    993     num_stack_levels=num_stack_levels,
    994     hyperparameters=hyperparameters,
    995     core_kwargs=core_kwargs,
    996     aux_kwargs=aux_kwargs,
    997     time_limit=time_limit,
    998     infer_limit=infer_limit,
    999     infer_limit_batch_size=infer_limit_batch_size,
   1000     verbosity=verbosity,
   1001     use_bag_holdout=use_bag_holdout,
   1002 )
   1003 self._set_post_fit_vars()
   1005 self._post_fit(
   1006     keep_only_best=kwargs["keep_only_best"],
   1007     refit_full=kwargs["refit_full"],
   (…)
   1012     infer_limit=infer_limit,
   1013 )

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/
 ↪tabular/learner/abstract_learner.py:159, in AbstractTabularLearner.fit(self,␣
 ↪X, X_val, **kwargs)
    157     raise AssertionError("Learner is already fit.")
    158 self._validate_fit_input(X=X, X_val=X_val, **kwargs)
--> 159 return self._fit(X=X, X_val=X_val, **kwargs)

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/
 ↪tabular/learner/default_learner.py:157, in DefaultLearner._fit(self, X, X_val␣
 ↪X_unlabeled, holdout_frac, num_bag_folds, num_bag_sets, time_limit,␣
 ↪infer_limit, infer_limit_batch_size, verbosity, **trainer_fit_kwargs)
    154     self.eval_metric = trainer.eval_metric
    156 self.save()
--> 157 trainer.fit(
    158     X=X,
    159     y=y,
```

```
    160     X_val=X_val,
    161     y_val=y_val,
    162     X_unlabeled=X_unlabeled,
    163     holdout_frac=holdout_frac,
    164     time_limit=time_limit_trainer,
    165     infer_limit=infer_limit,
    166     infer_limit_batch_size=infer_limit_batch_size,
    167     groups=groups,
    168     **trainer_fit_kwargs,
    169 )
    170 self.save_trainer(trainer=trainer)
    171 time_end = time.time()

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/
 ↪tabular/trainer/auto_trainer.py:114, in AutoTrainer.fit(self, X, y,␣
 ↪hyperparameters, X_val, y_val, X_unlabeled, holdout_frac, num_stack_levels,␣
 ↪core_kwargs, aux_kwargs, time_limit, infer_limit, infer_limit_batch_size,␣
 ↪use_bag_holdout, groups, **kwargs)
    111 log_str += "}"
    112 logger.log(20, log_str)
--> 114 self._train_multi_and_ensemble(
    115     X=X,
    116     y=y,
    117     X_val=X_val,
    118     y_val=y_val,
    119     X_unlabeled=X_unlabeled,
    120     hyperparameters=hyperparameters,
    121     num_stack_levels=num_stack_levels,
    122     time_limit=time_limit,
    123     core_kwargs=core_kwargs,
    124     aux_kwargs=aux_kwargs,
    125     infer_limit=infer_limit,
    126     infer_limit_batch_size=infer_limit_batch_size,
    127     groups=groups,
    128 )

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor
 ↪trainer/abstract_trainer.py:2371, in AbstractTrainer.
 ↪_train_multi_and_ensemble(self, X, y, X_val, y_val, hyperparameters,␣
 ↪X_unlabeled, num_stack_levels, time_limit, groups, **kwargs)
   2369     self._num_rows_val = len(X_val)
   2370 self._num_cols_train = len(list(X.columns))
-> 2371 model_names_fit = self.train_multi_levels(
   2372     X,
   2373     y,
   2374     hyperparameters=hyperparameters,
   2375     X_val=X_val,
   2376     y_val=y_val,
   2377     X_unlabeled=X_unlabeled,
```

```
   2378     level_start=1,
   2379     level_end=num_stack_levels + 1,
   2380     time_limit=time_limit,
   2381     **kwargs,
   2382 )
   2383 if len(self.get_model_names()) == 0:
   2384     raise ValueError("AutoGluon did not successfully train any models")

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:395, in AbstractTrainer.train_multi_levels(self,␣
 ↪X, y, hyperparameters, X_val, y_val, X_unlabeled, base_model_names,␣
 ↪core_kwargs, aux_kwargs, level_start, level_end, time_limit, name_suffix,␣
 ↪relative_stack, level_time_modifier, infer_limit, infer_limit_batch_size)
    393         core_kwargs_level["time_limit"] = core_kwargs_level.
 ↪get("time_limit", time_limit_core)
    394         aux_kwargs_level["time_limit"] = aux_kwargs_level.
 ↪get("time_limit", time_limit_aux)
--> 395     base_model_names, aux_models = self.stack_new_level(
    396         X=X,
    397         y=y,
    398         X_val=X_val,
    399         y_val=y_val,
    400         X_unlabeled=X_unlabeled,
    401         models=hyperparameters,
    402         level=level,
    403         base_model_names=base_model_names,
    404         core_kwargs=core_kwargs_level,
    405         aux_kwargs=aux_kwargs_level,
    406         name_suffix=name_suffix,
    407         infer_limit=infer_limit,
    408         infer_limit_batch_size=infer_limit_batch_size,
    409     )
    410     model_names_fit += base_model_names + aux_models
    411 if self.model_best is None and len(model_names_fit) != 0:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:539, in AbstractTrainer.stack_new_level(self, X,␣
 ↪y, models, X_val, y_val, X_unlabeled, level, base_model_names, core_kwargs,␣
 ↪aux_kwargs, name_suffix, infer_limit, infer_limit_batch_size)
    537     core_kwargs["name_suffix"] = core_kwargs.get("name_suffix", "") +␣
 ↪name_suffix
    538     aux_kwargs["name_suffix"] = aux_kwargs.get("name_suffix", "") +␣
 ↪name_suffix
--> 539 core_models = self.stack_new_level_core(
    540     X=X,
    541     y=y,
    542     X_val=X_val,
    543     y_val=y_val,
    544     X_unlabeled=X_unlabeled,
```

```
    545     models=models,
    546     level=level,
    547     infer_limit=infer_limit,
    548     infer_limit_batch_size=infer_limit_batch_size,
    549     base_model_names=base_model_names,
    550     **core_kwargs,
    551 )
    553 if X_val is None:
    554     aux_models = self.stack_new_level_aux(
    555         X=X, y=y, base_model_names=core_models, level=level + 1,␣
 ↪infer_limit=infer_limit, infer_limit_batch_size=infer_limit_batch_size,␣
 ↪**aux_kwargs
    556     )

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor/
 ↪trainer/abstract_trainer.py:673, in AbstractTrainer.stack_new_level_core(self ␣
 ↪X, y, models, X_val, y_val, X_unlabeled, level, base_model_names, stack_name, ,
 ↪ag_args, ag_args_fit, ag_args_ensemble, included_model_types,␣
 ↪excluded_model_types, ensemble_type, name_suffix, get_models_func, refit_full ␣
 ↪infer_limit, infer_limit_batch_size, **kwargs)
    670 fit_kwargs = dict(num_classes=self.num_classes)
    672 # FIXME: TODO: v0.1 X_unlabeled isn't cached so it won't be available␣
 ↪during refit_full or fit_extra.
--> 673 return self._train_multi(
    674     X=X_init,
    675     y=y,
    676     X_val=X_val,
    677     y_val=y_val,
    678     X_unlabeled=X_unlabeled,
    679     models=models,
    680     level=level,
    681     stack_name=stack_name,
    682     compute_score=compute_score,
    683     fit_kwargs=fit_kwargs,
    684     **kwargs,
    685 )

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor/
 ↪trainer/abstract_trainer.py:2321, in AbstractTrainer._train_multi(self, X, y, ,
 ↪models, hyperparameter_tune_kwargs, feature_prune_kwargs, k_fold, n_repeats,␣
 ↪n_repeat_start, time_limit, **kwargs)
   2319 if n_repeat_start == 0:
   2320     time_start = time.time()
-> 2321     model_names_trained = self._train_multi_initial(
   2322         X=X,
   2323         y=y,
   2324         models=models,
   2325         k_fold=k_fold,
   2326         n_repeats=n_repeats_initial,
   2327         hyperparameter_tune_kwargs=hyperparameter_tune_kwargs,
```

```
   2328         feature_prune_kwargs=feature_prune_kwargs,
   2329         time_limit=time_limit,
   2330         **kwargs,
   2331     )
   2332     n_repeat_start = n_repeats_initial
   2333     if time_limit is not None:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:2170, in AbstractTrainer.
 ↪_train_multi_initial(self, X, y, models, k_fold, n_repeats,␣
 ↪hyperparameter_tune_kwargs, time_limit, feature_prune_kwargs, **kwargs)
   2168 else:
   2169     time_ratio = hpo_time_ratio if hpo_enabled else 1
-> 2170     models = self._train_multi_fold(
   2171         models=models,
   2172         hyperparameter_tune_kwargs=hyperparameter_tune_kwargs,
   2173         k_fold_start=0,
   2174         k_fold_end=k_fold,
   2175         n_repeats=n_repeats,
   2176         n_repeat_start=0,
   2177         time_limit=time_limit,
   2178         time_split=time_split,
   2179         time_ratio=time_ratio,
   2180         **fit_args,
   2181     )
   2183 multi_fold_time_elapsed = time.time() - multi_fold_time_start
   2184 if time_limit is not None:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:2278, in AbstractTrainer._train_multi_fold(self,␣
 ↪X, y, models, time_limit, time_split, time_ratio, hyperparameter_tune_kwargs,␣
 ↪**kwargs)
   2276         time_start_model = time.time()
   2277         time_left = time_limit - (time_start_model - time_start)
-> 2278 model_name_trained_lst = self._train_single_full(
   2279     X, y, model, time_limit=time_left,␣
 ↪hyperparameter_tune_kwargs=hyperparameter_tune_kwargs_model, **kwargs
   2280 )
   2282 if self.low_memory:
   2283     del model

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:2051, in AbstractTrainer._train_single_full(self,␣
 ↪X, y, model, X_unlabeled, X_val, y_val, X_pseudo, y_pseudo, feature_prune,␣
 ↪hyperparameter_tune_kwargs, stack_name, k_fold, k_fold_start, k_fold_end,␣
 ↪n_repeats, n_repeat_start, level, time_limit, fit_kwargs, compute_score,␣
 ↪total_resources, **kwargs)
   2047         bagged_model_fit_kwargs = self._get_bagged_model_fit_kwargs(
   2048             k_fold=k_fold, k_fold_start=k_fold_start,␣
 ↪k_fold_end=k_fold_end, n_repeats=n_repeats, n_repeat_start=n_repeat_start
```

```
   2049         )
   2050         model_fit_kwargs.update(bagged_model_fit_kwargs)
-> 2051     model_names_trained = self._train_and_save(
   2052         X=X,
   2053         y=y,
   2054         model=model,
   2055         X_val=X_val,
   2056         y_val=y_val,
   2057         X_unlabeled=X_unlabeled,
   2058         stack_name=stack_name,
   2059         level=level,
   2060         compute_score=compute_score,
   2061         total_resources=total_resources,
   2062         **model_fit_kwargs,
   2063     )
   2064 self.save()
   2065 return model_names_trained

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:1733, in AbstractTrainer._train_and_save(self, X, 
 ↪y, model, X_val, y_val, stack_name, level, compute_score, total_resources,␣
 ↪**model_fit_kwargs)
   1731     model = self._train_single(X_w_pseudo, y_w_pseudo, model, X_val,␣
 ↪y_val, **model_fit_kwargs)
   1732 else:
-> 1733     model = self._train_single(X, y, model, X_val, y_val,␣
 ↪total_resources=total_resources, **model_fit_kwargs)
   1735 fit_end_time = time.time()
   1736 if self.weight_evaluation:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪trainer/abstract_trainer.py:1684, in AbstractTrainer._train_single(self, X, y ␣
 ↪model, X_val, y_val, total_resources, **model_fit_kwargs)
   1679 def _train_single(self, X, y, model: AbstractModel, X_val=None,␣
 ↪y_val=None, total_resources=None, **model_fit_kwargs) -> AbstractModel:
   1680     """
   1681     Trains model but does not add the trained model to this Trainer.
   1682     Returns trained model object.
   1683     """
-> 1684     model = model.fit(X=X, y=y, X_val=X_val, y_val=y_val,␣
 ↪total_resources=total_resources, **model_fit_kwargs)
   1685     return model

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪models/abstract/abstract_model.py:829, in AbstractModel.fit(self, **kwargs)
    827 self.validate_fit_resources(**kwargs)
    828 self._validate_fit_memory_usage(**kwargs)
--> 829 out = self._fit(**kwargs)
```

```
    830 if out is None:
    831     out = self

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪models/ensemble/stacker_ensemble_model.py:169, in StackerEnsembleModel.
 ↪_fit(self, X, y, compute_base_preds, time_limit, **kwargs)
    167 if time_limit is not None:
    168     time_limit = time_limit - (time.time() - start_time)
--> 169 return super()._fit(X=X, y=y, time_limit=time_limit, **kwargs)

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪models/ensemble/bagged_ensemble_model.py:266, in BaggedEnsembleModel.
 ↪_fit(self, X, y, X_val, y_val, X_pseudo, y_pseudo, k_fold, k_fold_start,␣
 ↪k_fold_end, n_repeats, n_repeat_start, groups, _skip_oof, **kwargs)
    264         # Reserve time for final refit model
    265         kwargs["time_limit"] = kwargs["time_limit"] * folds_to_fit /␣
 ↪(folds_to_fit + 1.2)
--> 266 self._fit_folds(
    267     X=X,
    268     y=y,
    269     model_base=model_base,
    270     X_pseudo=X_pseudo,
    271     y_pseudo=y_pseudo,
    272     k_fold=k_fold,
    273     k_fold_start=k_fold_start,
    274     k_fold_end=k_fold_end,
    275     n_repeats=n_repeats,
    276     n_repeat_start=n_repeat_start,
    277     save_folds=save_bag_folds,
    278     groups=groups,
    279     **kwargs,
    280 )
    281 # FIXME: Cleanup self
    282 # FIXME: Support `can_refit_full=False` models
    283 if refit_folds:

File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor /
 ↪models/ensemble/bagged_ensemble_model.py:592, in BaggedEnsembleModel.
 ↪_fit_folds(self, X, y, model_base, X_pseudo, y_pseudo, k_fold, k_fold_start,␣
 ↪k_fold_end, n_repeats, n_repeat_start, time_limit, sample_weight, save_folds,␣
 ↪groups, num_cpus, num_gpus, **kwargs)
    590 for fold_fit_args in fold_fit_args_list:
    591     fold_fitting_strategy.schedule_fold_model_fit(**fold_fit_args)
--> 592 fold_fitting_strategy.after_all_folds_scheduled()
    594 for model in models:
    595     # No need to add child times or save child here as this already␣
 ↪occurred in the fold_fitting_strategy
    596     self.add_child(model=model, add_child_times=False)
```

```
File /opt/homebrew/anaconda3/envs/ag/lib/python3.10/site-packages/autogluon/cor/
 ↪models/ensemble/fold_fitting_strategy.py:538, in ParallelFoldFittingStrategy.
 ↪after_all_folds_scheduled(self)
    536 unfinished = job_refs
    537 while unfinished:
--> 538     finished, unfinished = self.ray.wait(unfinished, num_returns=1)
    539     finished = finished[0]
    540     try:

File ~/.local/lib/python3.10/site-packages/ray/_private/client_mode_hook.py:105 ␣
 ↪in client_mode_hook.<locals>.wrapper(*args, **kwargs)
    103     if func.__name__ != "init" or is_client_mode_enabled_by_default:
    104         return getattr(ray, func.__name__)(*args, **kwargs)
--> 105 return func(*args, **kwargs)

File ~/.local/lib/python3.10/site-packages/ray/_private/worker.py:2578, in␣
 ↪wait(object_refs, num_returns, timeout, fetch_local)
   2576 timeout = timeout if timeout is not None else 10**6
   2577 timeout_milliseconds = int(timeout * 1000)
-> 2578 ready_ids, remaining_ids = worker.core_worker.wait(
   2579     object_refs,
   2580     num_returns,
   2581     timeout_milliseconds,
   2582     worker.current_task_id,
   2583     fetch_local,
   2584 )
   2585 return ready_ids, remaining_ids

File python/ray/_raylet.pyx:1833, in ray._raylet.CoreWorker.wait()

File python/ray/_raylet.pyx:199, in ray._raylet.check_status()

KeyboardInterrupt:
```

[ ]:
```
loc = "C"
print(f"Training model for location {loc}...")
predictor = TabularPredictor(label=label, eval_metric=metric,␣
 ↪path=f"AutogluonModels/{new_filename}_{loc}").
 ↪fit(train_data[train_data["location"] == loc], time_limit=time_limit,␣
 ↪presets=presets)
predictors[2] = predictor
```

2 Submit

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

train_data_with_dates = TabularDataset('X_train_raw.csv')
train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])

test_data = TabularDataset('X_test_raw.csv')
test_data["ds"] = pd.to_datetime(test_data["ds"])
#test_data
```

[ ]:
```
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
test_data_merged = pd.merge(test_data, test_ids, how="inner", right_on=["time",␣
 ↪"location"], left_on=["ds", "location"])

#test_data_merged
```

[ ]:
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in test_data.groupby('location'):
    i = location_map[loc]
    subset = test_data_merged[test_data_merged["location"] == loc].
 ↪reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)
```

[ ]:
```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
 ↪y='y', ax=ax, label="train data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # title
```

```
    ax.set_title(f"Predictions for location {loc}")
```

[ ]:
```
# concatenate predictions
submissions_df = pd.concat(predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

[ ]:
```
# Save the submission DataFrame to submissions folder, create new name based on␣
 ↪last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),␣
 ↪index=False)
```

[ ]:
```
# save this notebook to submissions folder
import subprocess
import os
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
 ↪ipynb"])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/Users/jorgensandhaug/Desktop/tdt4173/data/autogluon_each_location.ipynb Cell 1␣
 ↪line 4
      <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#X23sZmlsZQ%3D%3D?line=1'>2</a> import subproces
      <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#X23sZmlsZQ%3D%3D?line=2'>3</a> import os
----> <a href='vscode-notebook-cell:/Users/jorgensandhaug/Desktop/tdt4173/data/
 ↪autogluon_each_location.ipynb#X23sZmlsZQ%3D%3D?line=3'>4</a> subprocess.
 ↪run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.ipynb" )

NameError: name 'new_filename' is not defined
```